Getting Hands-On Experience with Silq

All code snippets can be copied from https://silq.ethz.ch/downloads/silq-exercise-snippets.slq.

Task 1: Writing an Oracle for Grover's Algorithm

Introduction. Grover's algorithm is a widely known quantum algorithm. For a given oracle function $f: \{0, ...2^n - 1\} \to \{0, 1\}$ mapping n-bit unsigned integers to booleans, it finds the input $w^\star$ for which $f(w^\star) = 1$. For simplicity, we assume there exists only one such $w^\star$.

Silq Implementation. On the next page, we provide a simple Silq implementation of Grover's algorithm, including a dummy oracle function. Running this code should return 2.

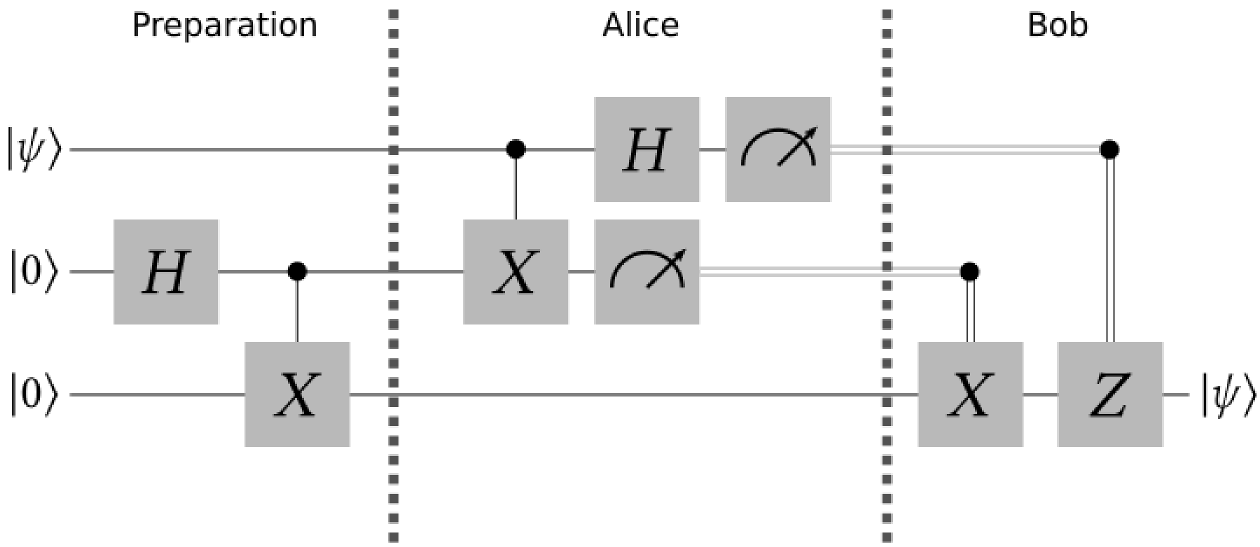
Task. Create a new oracle function to find x between 10 and 20 with x % 2 = 1 and x % 5 = 4. Hint: The only solution satisfying these constraints is 19.

Bonus Task. Try to implement the same oracle without Silq's automatic uncomputation. Hint: If `x` was computed by `x:= a+b`, Silq allows you to uncompute it using `forget(x = a+b)`.

```
def groverDiffusion[n:!ℕ](cand:uint[n]) mfree: uint[n] {
   for k in [0..n) { cand[k] := H(cand[k]); }
   if cand!=0 { phase(π); }
   for k in [0..n) { cand[k] := H(cand[k]); }
   return cand;
}

def grover[n:!ℕ](f:uint[n] !→ lifted 𝔹){
   nIterations:=floor(π/(4·asin(2^(-n/2))));
   cand:=0:uint[n];
   for k in [0..n) { cand[k] := H(cand[k]); }

   for k in [0..nIterations){
       b := f(cand);
       if b { phase(π); }
       forget(b = f(cand));
       cand:=groverDiffusion(cand);
   }
   return measure(cand);
}

def dummy_oracle(x:uint[5]) lifted{
   // TODO: complete the oracle
   return x == 2; // a simple oracle that checks if the input equals 2
}

def main() { // run grover on dummy oracle
   return grover(dummy_oracle);
}
```

Task 2: Implementing Quantum Teleportation

Introduction. Quantum teleportation allows a sender to transfer an unknown quantum state to a receiver by transmitting classical information. This is a circuit description of quantum teleportation, where double wires indicate wires carrying classical information:



Task. Implement quantum teleportation in Silq by completing the following program.

```
// generate state |Φ⁺⟩
def preparation(){
   // TODO: COMPLETE
}

def alice(ψ:𝔹,a:𝔹){
   // TODO: COMPLETE
}

def bob(measured_a:!𝔹,measured_ψ:!𝔹,b:𝔹){
   // TODO: COMPLETE
}

def teleportation(ψ:𝔹){
   (a,b) := preparation();
   (measured_a, measured_ψ) := alice(ψ,a);
   ψ := bob(measured_a, measured_ψ, b);
   return ψ;
}

def main(){ // tests checking teleportation works correctly
   assert(measure(teleportation(0:𝔹))==0); // teleport 0
   assert(measure(H(teleportation(H(0:𝔹))))==0); // teleport |+⟩
   assert(measure(teleportation(1:𝔹))==1); // teleport 1
   assert(measure(H(teleportation(H(1:𝔹))))==1); // teleport |-⟩
}
```

Solution 1

We can replace the dummy oracle by this function:

```
def dummy_oracle(x:uint[5]) lifted{
   return
       x % 2 == 1 &&
       10 <= x &&
       x <= 20 &&
       x % 5 == 4;
}
```

Making uncomputation explicit is highly inconvienent:

```
def dummy_oracle_explicit(x:uint[5]) lifted{
   x_mod_2 := x % 2;
   x_mod_2_eq_1 := x_mod_2 == 1;

   ten_leq_x := 10 <= x;

   x_leq_20 := x <= 20;

   x_mod_5 := x % 5;
   x_mod_5_eq_4 := x_mod_5 == 4;

   two := x_mod_2_eq_1 && ten_leq_x;
   three := two && x_leq_20;
   four := three && x_mod_5_eq_4;

   // uncomputation

   forget(three = two && x_leq_20);
   forget(two = x_mod_2_eq_1 && ten_leq_x);

   forget(x_mod_5_eq_4 = x_mod_5 == 4);
   forget(x_mod_5 = x % 5);

   forget(x_leq_20 = x <= 20);
   forget(ten_leq_x = 10 <= x);

   forget(x_mod_2_eq_1 = x_mod_2 == 1);
   forget(x_mod_2 = x % 2);

   return four;
}
```

Solution 2

This is a straight-forward implementation of the shown quantum circuit:

```
// generate state |Φ⁺⟩
def preparation(){
   // prepare a in state |+⟩
   a := H(0:𝔹);

   // prepare b to obtain state |00⟩ + |11⟩ (ignoring normalization)
   b := 0:𝔹;
   if a {
       b := X(b);
   }

   return (a,b);
}

def alice(ψ:𝔹,a:𝔹){
   if ψ {
       a := X(a);
   }

   ψ := H(ψ);

   return (measure(a), measure(ψ));
}

def bob(measured_a:!𝔹,measured_ψ:!𝔹,b:𝔹){
   if measured_a {
       b := X(b);
   }

   if measured_ψ {
       b := Z(b);
   }

   return b;
}

def teleportation(ψ:𝔹){
   (a,b) := preparation();
   (measured_a, measured_ψ) := alice(ψ,a);
   ψ := bob(measured_a, measured_ψ, b);
   return ψ;
}

def main(){ // tests checking teleportation works correctly
   assert(measure(teleportation(0:𝔹))==0); // teleport |0⟩
   assert(measure(H(teleportation(H(0:𝔹))))==0); // teleport |+⟩
   assert(measure(teleportation(1:𝔹))==1); // teleport |1⟩
   assert(measure(H(teleportation(H(1:𝔹))))==1); // teleport |-⟩
}
```